In the Claims

1. (Currently amended) A method of automatically configuring an architecture of a target microprocessor of a first type, comprising the steps of:

(a) a processor generation tool taking, as an input, executable code for an existing microprocessor of a second type differing from said first type;

(b) the said executable code implementing a software application ~~for~~ requiring enhanced performance ~~that is required~~ over that attainable by direct execution by a microprocessor of the second type;

(c) the processor generation tool ~~automatically~~ adapting the instruction set of the microprocessor of the first type at design time, such that it is optimized for the execution of the software application represented by the said executable code

(d) the processor generation tool outputting a hardware description of a microprocessor of the first type that implements the adapted instruction set which differs from that of the second type; ~~and~~

(e) a code generation tool outputting executable code for the first type of microprocessor by using executable code for the microprocessor of the second type as input and translating that code so that identical program behavior is provided but with enhanced performance enabled by use of the adapted instruction set of the first type;

(f) the microprocessor of the first type with its adapted instruction set being used as a coprocessor in a system for a microprocessor of the second type;

(g) a number of individual software functions in the executable code being marked for translation and execution on said coprocessor;

(h) the said executable code being modified by a tool so that function calls to those functions marked for translation cause an equivalent translated function to be executed on said coprocessor; and

(i) the modified executable code being executed on the microprocessor of the second type.

Claims 2 to 5 (Cancelled)

6. (Currently amended) The method according to claim [[5]] 1 whereby an original executable image is automatically modified so that function calls to those translated functions cause an equivalent function to be executed on the coprocessor.

7. (original) The method according to claim 6 whereby a coprocessor initiation involves the transfer of register state from the host processor to the coprocessor.

8. (original) The method according to claim 7 whereby completion of a function on the coprocessor causes the transfer of register state from the coprocessor to the host processor.

9. (original) The method according to claim 7 whereby completion of a function on the coprocessor causes the transfer of memory state from the coprocessor to the host processor.

10. (Previously presented) The method according to claim 1 whereby the configured architecture generated is designed to execute parts of the executable with higher performance than can be achieved with a host processor.

11. (Previously presented) The method according to claim 10 whereby the higher performance is obtained by the execution of more operations in parallel than is achieved with the host processor.

12. (Previously presented) The method according to claim 1 whereby the architecture generated is designed to execute parts of the executable with lower power consumption than can be achieved with a host processor.

13. (original) The method according to claim 1 whereby the executable code is translated into the instruction set of the configured processor.

14. (Previously presented) The method according to claim 13 whereby each

instruction in an executable image is translated into one or more basic operations.

15 (original) The method according to claim 14 whereby each of these operations may be performed by a particular execution unit that is present in the configured processor.

16. (Previously presented) The method according to claim 15 whereby a register file is present as an execution unit in the architecture and explicit operations to read and write the register file is generated as part of the translation.

17. (original) The method according to claim 16 whereby static register analysis may be used to eliminate unnecessary writes of registers.

18. (original) The method according to claim 17 whereby code is subdivided into atomically executed blocks.

19. (original) The method according to claim 18 whereby each atomically executed block reproduces the operations of the corresponding host code.

20. (original) The method according to claim 19 whereby the state of live registers at the end of the atomic block execution is identical to that obtained from execution on the host processor.

21. (original) The method according to claim 19 whereby the state of the memory at the end of the atomic block execution is identical to that obtained from execution on the host processor.

22. (original) The method according to claim 1 whereby breakpoints may be set on the configured architecture.

23. (original) The method according to claim 22 whereby the breakpoints may be specified using the addresses of instructions in the original executable.

24. (Previously presented) The method according to claim 23 whereby the nearest preceding instruction for which state can be synchronized on the configured processor is determined when the breakpoint is set.

25. (original) The method according to claim 24 whereby the configured processor contains a mechanism to determine the equivalent target instruction address for a host instruction address.

26. (original) The method according to claim 25 whereby the configured processor contains hardware to cause a breakpoint halt on the required address that prevents any side effects caused by sequentially later instructions.

27. (Previously presented) The method according to claim 26 whereby upon detection of a breakpoint execution can be continued on the host processor from the synchronized address until the point of the actual breakpoint.

28. (original) The method according to claim 23 whereby the breakpoint address is determined by decoding the data stream on the debug interface to the host processor.

29. (original) The method according to claim 1 whereby certain host processor instruction addresses may be converted to target processor addresses while the system is running.

30. (original) The method according to claim 29 whereby a hashing table is maintained in memory to perform a mapping of certain host processor instruction addresses to target addresses.

31. (original) The method according to claim 30 whereby mapping information may be interleaved with the machine code for the target processor.

32. (original) The method according to claim 31 whereby the state of certain bits within each word of a table are used to indicate whether the information represents an address mapping entry or target machine code.

33. (original) The method according to claim 1 whereby function calls in the executable code may be replaced with uses of particular hardware blocks in the configured processor.

34. (original) The method according to claim 33 whereby the input parameters to the software function correspond to the operands supplied to the corresponding hardware unit.

35. (original) The method according to claim 34 whereby the reference parameters and return result from a software function correspond to the results generated by the corresponding hardware unit.

36. (Previously presented) The method according to claim 33 whereby the original software implementation may be used as a behavioral model for hardware for the purposes of simulation.

37. (Previously presented) The method according to claim 1 whereby an instruction set translator converts instructions from an executable image into behaviorally equivalent operations that are mapped to a target processor.

38. (Currently amended) The method according to claim [[4]] 1 whereby the coprocessor contains one or more cache memories.

39. (original) The method according to claim 38 whereby the coprocessor and host processor communicate via a system bus or a generic coprocessor interface on the host processor.

40. (original) The method according to claim 39 whereby the host processor services memory access requests from the coprocessor while the coprocessor is operational.

41. (original) The method according to claim 40 whereby the coprocessor is able to flush its caches of all modified data when the end of function execution is reached.

42. (original) The method according to claim 39 whereby a copy of some virtual to physical page mappings are maintained by the coprocessor.

43. (Currently amended) A first microprocessor automatically configured by a processor generation tool, in which the first microprocessor using the method as defined in claim 1, stores, and operates using, an instruction set ~~that has been automatically~~ adapted, in dependence on the requirements of executable code of a second existing microprocessor of a type differing from that of said the first microprocessor, by the processor generation tool.